NETWORK WORKING GROUP
Request for Comments #114
NIC 5823

Abhay K. Bhushan
MIT Project MAC
16 April 1971

A FILE TRANSFER PROTOCOL

[Categories: D.4, D.5, D.7]

I. Introduction

Computer network usage may be divided into two broad categories -- direct and indirect. Direct usage implies that you, the network user, are "logged" into a remote host and use it as a local user. You interact with the remote system via a terminal (teletypewriter, graphics console) or a computer. Differences in terminal characteristics are handled by host system programs, in accordance with standard protocols (such as TELNET (RFC 97) for teletypewriter communications, NETRJS (RFC 88) for remote job entry). You, however, have to know the different conventions of remote systems, in order to use them.

Indirect usage, by contrast, does not require that you explicitly log into a remote system or even know how to "use" the remote system. An intermediate process makes most of the differences in commands and conventions invisible to you. For example, you need only know a standard set of network file transfer commands for your local system in order to utilize remote file system. This assumes the existence of a network file transfer process at each host cooperating via a common protocol.

Indirect use is not limited to file transfers. It may include execution of programs in remote hosts and the transfer of

core images. The extended file transfer protocol would facilitate the exchange of programs and data between computers, the use of storage and file handling capabilities of other computers (possibly including the trillion-bit store data computer), and have programs in remote hosts operate on your input and return an output.

The protocol described herein has been developed for immediate implementation on two hosts at MIT, the GE645/Multics and the PDP-1Ø/DM/CG-ITS (and possibly Harvard's PDP-1Ø). An interim version with limited capabilities is currently in the debugging stage.* Since our implementation involves two dissimiliar systems (Multics is a "service" system, ITS is not) with different file systems (Multics provides elaborate access controls, ITS provides none), we feel that the file transfer mechanisms proposed are generalizable. In addition, our specification reflects a consideration of other file systems on the network. We conducted a survey** of network host systems to determine the requirements and capabilities. This paper is a "first cut" at a protocol that will allow users at any host on the network to use the file system of every cooperating host.

---

* The interim version of the protocol, limited to transfer of ASCII files, was developed by Chander Ramchandani and Howard Brodie of Project MAC. The ideas of transactions, descriptors, error recovery, aborts, file headings and attributes, execution of programs, and use of data types, pathnames, and default mechanisms are new here. Howard Brodie and Neal Ryan have coded the interim protocol in the PDP-1Ø and the 645, respectively.

** The network system survey was conducted last fall by Howard Brodie of Project MAC, primarily by telephone.

II. Discussion

A few definitions are in order before the discussion of the protocol. A file is an ordered set consisting of computer instructions and/or data. A file can be of arbitrary length.* A named file is uniquely identified in a system by its file name and directory name. The directory name may be the name of a physical directory or it may be the name of a physical device. An example of physical directory name is owner's project-programmer number and an example of physical device name is tape number.

A file may or may not have access controls associated with it. The access controls designate the users' access privileges. In the absence of access controls, the files cannot be protected from accidental or unauthorized usage.

A principal objective of the protocol is to promote the indirect use of computers on the network. Therefore, the user or his program should have a simple and uniform interface to the file systems on the network and be shielded from the variations in file and storage systems of different host computers. This is achieved by the existence of a standard protocol in each host.

Criteria by which a user-level protocol may be judged were described by Mealy in RFC 91, as involving the notion of logical records, ability to access files without program modifications, and implementatability. I would add to these efficiency, extendability, adaptability, and provision of error-recovery

---

* PDP-1Ø Reference Handbook, page 306.

mechanisms.

The attempt in this specification has been to enable the reliable transfer of network ASCII (7-bit ASCII in 8-bit field with leftmost bit zero) as well as "binary" data files with relative ease. The use of other character codes, such as EBCDIC, and variously formatted data (decimal, octal, ASCII characters packed differently) is facilitated by inclusion of data types in descriptor headings. An alternative mechanism for defining data is also available in the form of attributes in file headings. The format control characters reserved for the syntax of this protocol have identical code representation in ASCII and EBCDIC. (These character are SOH, STX, ETX, DC1, DC2, DC3, US, RS, GS, and FS.)

The notion of messages (the physical blocks of data communicated between NCP's) is suppressed herein and that of "logical" records and transactions is emphasized. The data passed by the NCP is parsed into logical blocks by use of simple descriptors (code and count mechanisms) as described in Section III. The alternative to count is fixed length blocks or standard end-of-file characters (scan data stream). Both seem less desirable than count.

The cooperating processes may be "daemon" processes which "listen" to agreed-upon sockets, and follow the initial connection protocol much in the same way as a "logger" does. We recommend using a single full-duplex connection for the exchange

of both data and control information,* and using CLS to achieve synchronization when necessary (a CLS is not transmitted until a RFNM is received).

The user may be identified by having the using process send at the start of the connection the user's name information (either passed on by user or known to the using system).** This user name information (a sequence of standard ASCII characters), along with the host number (known to the NCP), positively identifies the user to the serving process.

At present, more elaborate access control mechanisms, such as passwords, are not suggested. The user, however, will have the security and protection provided by the serving system. The serving host, if it has access controls, can prevent unprivileged access by users from other host sites. It is up to the using host to prevent its own users from violating access rules.

The files in a file system are identified by a pathname, similar to the labels described in RFC 76 (Bouknight, Madden and Grossman). The pathname contains the essential information

---

* We considered using two full-duplex links, one for control information, the other for data. The use of a separate control link between the cooperating processes would simplify aborts, error recoveries and synchronization. The synchronization function may alternatively be performed by closing the connection (in the middle of a transaction sequence) and reopening it with an abort message. (The use of INR and INS transmitted via the NCP control link has problems as mentioned by Kalin in RFC 103.) We prefer the latter approach.

** Identifying users through use of socket numbers is not practical, as unique user identification numbers have not been implemented, and file systems identify users by name, not number.

regarding the storage and retrieval of data. In order to facilitate use, default options should be provided. For example, the main file directory on disk would be the default on the PDP-1Ø/ITS, and a pool directory would be the default on Multics.

The file to be transferred may be a complete file or may consist of smaller records. It may or may not have a heading. A heading should contain ASCII or EBCDIC characters defining file attributes. The file attributes could be some simple agreed-upon types or they could be described in a data reconfiguration or interpretation language similar to that described in RFC 83 (Anderson, Haslern and Heffner), or a combination.

The protocol does not restrict the nature of data in the file. For example, a file could contain ASCII text, binary core image, graphics data or any other type of data. The protocol includes an "execute" request for files that are programs. This is intended to facilitate the execution of programs and subroutines in remote host computers.*

---

* This subject is considered in detail by Bob Metcalfe in a forthcoming paper.

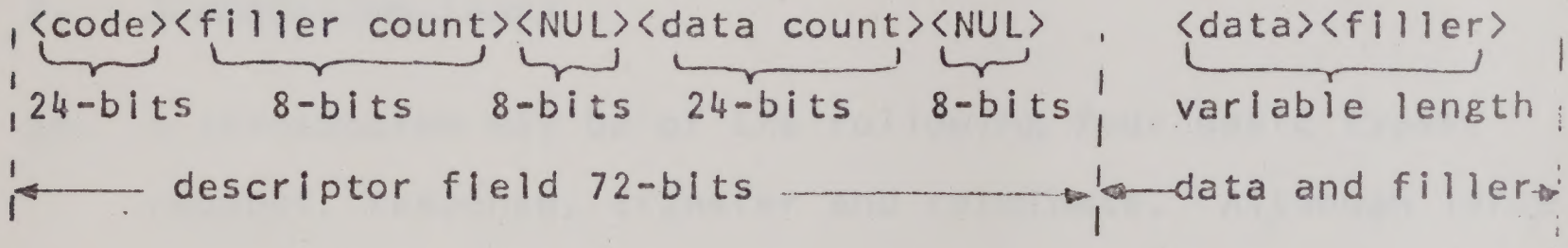
III. SPECIFICATIONS

1. Transactions

1A. The protocol is transaction-oriented. A transaction is defined to be an entity of information communicated between cooperating processes.

1B. Syntax

A transaction has three fields, a 72-bit descriptor field and variable length (including zero) data and filler fields, as shown below. The total length of a transaction is (72 + data + filler) bits.



1C. Semantics

The code field has three 8-bit bytes. The first byte is interpreted as transaction type, the second byte as data type and the third byte as extension of data type.

The filler count is a binary count of bits used as "filler" (i.e., not information) at the end of a transaction.* As the length of the filler count field *

---

* Filler bits may be necessary as particular implementations of NCP's may not allow the free communication of bits. Instead the NCP'S may only accept bytes, as suggested in RFC 1Ø2. The filler count is needed to determine the boundary between transactions.

is 8-bits, the number of bits of filler shall not exceed 255 bits.

The data count is a binary count of the number of data (i.e., information) bits in the data field, not including filler bits. The number of data bits is limited to $(2^{24} - 1)$, as there are 24 bits in the data count field.

The NUL bytes are inserted primarily as fillers in the descriptor field and allow the count information to appear at convenient word boundaries for different word length machines.*

2. Transaction Types

2A. A transaction may be of the following four basic types: request, response, transfer and terminate. Although large number of request and transfer types are defined, implementation of a subset is specifically permitted. Host computers, on which a particular transaction type is not implemented, may refuse to accept that transaction by responding with an unsuccessful terminate.

The following transaction type codes are tentatively defined:

---

* 72-bits in descriptor field are convenient as 72 is the least common multiple of 6,8,9,18,24 and 36, the commonly encountered byte sizes on the ARPA network host computers.

Transaction Type		Transaction Type Code		
		ASCII	Octal	Hexidecimal
Request				
	Identify	I	111	49
	Retrieve	R	122	52
	Store	S	123	53
	Append	A	101	41
	Delete	D	104	44
	Rename	N	116	4E
	addname(Plus)	P	120	50
	deletename(Minus)	M	115	4D
	Lookup	L	114	4C
	Open	O	117	4F
	Close	C	103	43
	Execute *	E	105	45
Response				
	ready-to-receive (rr)	<	074	3C
	ready-to-send (rs)	>	076	3E
Transfer				
	complete_file	*	052	
	heading	#	043	23
	part_of_file	,	054	2C
	last_part	.	056	2E
Terminate				
	successful (pos.)	+	053	2B
	unsuccessful (neg.)	-	055	2D

---

* The execute request is intended to facilitate the indirect execution of programs and subroutines. However, this request in its present form may have only limited use. A subroutine or program mediation protocol would be required for broader use of the execute feature. Metcalfe considers this problem in a forthcoming paper.

2B. Syntax

In the following discussion US, RS, GS, FS, DC1, DC2, and DC3 are the ASCII characters, unit separator (octal Ø37), record separator (octal Ø36), group separator (octal Ø35), file separator (octal Ø34), device control 1 (octal Ø21), device control 2 (octal Ø22), and device control 3 (octal Ø23), respectively. These have an identical interpretation in EBCDIC.

2B.1 Requests

Identify, retrieve, store, append, delete, open, lookup and execute requests have the following data field:

<path name>

Rename request has the data field:

<path name> GS <name>

Addname and deletename requests have the data field:

<path name> GS <filenames>

where pathname,* name and filenames have the following syntax (expressed in BNF, the metalanguage of the ALGOL 6Ø report):

<pathname> ::= <device name>|<name>|<pathname>US<name>

<device name> ::= DC1<name>

---

* The pathname idea used in Multics is similar to that of labels in RFC 76 by Bouknight, Madden and Grossman.

<name> ::= <char> | <name> <char>

<char> ::= All 8-bit ASCII or EBCDIC characters except

US, RS, GS, FS, DC1, DC2, and DC3.

<filenames> ::= <name>|<filenames> RS <name>

The data type for the request transaction shall be either A (octal 101 for ASCII, or E (octal 105) for EBCDIC.*

Some examples of pathname are:

DC1 MT08

DC1 DSK 1.2 US Net<3> US J.Doe US Foo

udd US proj. US h,n/x US user US file

filename 1 filename 2

2B.2 Responses

The response transactions shall normally have an empty data field.

2B.3 Transfers

The data types defined in section 4 will govern the syntax of the data field in transfer transactions. No other syntactical restrictions exist.

2B.4 Terminates

The successful terminate shall normally have an empty data field. The unsuccessful terminate may have a data field

---

* We, however, urge the use of standard network ASCII.

defined by the data types A (octal 101) for ASCII, E (octal 105) for EBCDIC, or S (octal 123) for status.

A data type code of 'S' would imply byte oriented error return status codes in the data field. The following error return status codes are defined tentatively:

Error Code Meaning	Error Code		
	ASCII	Octal	Hexadecimal
Undefined error	U	125	55
Transaction type error	T	124	54
Syntax error	S	123	53
File search failed	F	106	46
Data type error	D	104	44
Access denied	A	101	41
Improper transaction sequence	I	111	49
Time-out error	O	117	4F
Error condition by system	E	105	45

2C. Semantics

2C.1 Requests

Requests are alway sent by using host. In absence of a device name or complete pathname, default options should be provided for all types of requests.

Identify request identifies the user as indicated by <pathname>.

Retrieve request achieves the transfer of file specified in <pathname> from serving to using host.

Store request achieves the transfer of file specified in <pathname> from using to serving host. Append request causes data to be added to file specified in pathname.

Rename request causes name of file specified in <pathname> to be replaced by name specified in <name>. Delete request causes file specified in <pathname> to be deleted. If an extra level of protection for delete is desired (such as the query 'Do you wish to delete file x?'), it is to be a local implementation option.

Addname and deletename requests cause names in <filenames> to be added or deleted to existing names of file specified in <pathname>. These requests are useful in systems such as Multics which allow multiple names to be associated with a file.

Lookup request achieves the transfer of attributes (such as date last modified, access list, etc) of file specified in <pathname>, instead of the file itself.

Open request does not cause a data transfer, instead file specified in <pathname> is "opened" for retrieve (read) or store (write). Subsequent requests are then treated as requests pertaining to the file that is opened till such a time that a close request is received.

Execute request achieves the execution of file specified in <pathname>, which must be an executable program. Upon

receipt of rr response, using host will transmit the necessary input data (parameters, arguments, etc). Upon completion of execution serving host will send the results to using host and terminate. *

2C.2 Response

Responses are always sent by serving host. The rr response indicates that serving host is ready to receive the file indicated in the preceeding request. The rs response indicates that the next transaction from serving host will be the transfer of file indicated in the preceeding request.

2C.3 Transfers

Transfers may be sent by either host. Transfer transactions indicate the transfer of file indicated by a request. Files can be transferred either as complete_file transactions or as part_of_file transactions followed by last_part transactions. The file may also have a heading transaction in the beginning. The syntax of a file, therefore, may be defined as:

<file> ::= <text> | <heading> <text>

<text> ::= <complete_file> | <parts> <last_part>

<parts> ::= <part_of_file> | <parts> <part_of_file>

---

* The exact manner in which the input and output are transmitted would depend on specific mediation conventions. Names of input and output files may be transmitted instead of data itself.

Headings may be used to communicate the attributes of files. The form of headings is not formally specified but is discussed in Section IV as possible extension to this protocol.

2C.4 Terminates

The successful terminate is always sent by serving host. It indicates to using host that serving host has been successful in serving the request and has gone to an initial state. Using host will then inform user that his request is successfully served, and go to an initial state.

The unsuccessful terminate may be sent by either host. It indicates that sender of the terminate is unable to (or does not not wish to) go through with the request. Both hosts will then go to their initial states. The using host will inform the user that his request was aborted. If any reasons for the unsuccessful terminate (either as text or as error return status codes) are received, these shall be communicated to the user.

3. Transaction Sequence

3A. The transaction sequence may be defined as an instance of file transfer, initiated by a request and ended by a

terminate.* The exact sequence in which transactions occur depends on the type of request. A transaction sequence may be aborted anytime by either host, as explained in Section 3C.

3B. Examples

The identify request doesn't require a response or terminate and constitutes a transaction sequence by itself.

Rename, delete, addname, deletename and open requests involve no data transfer but require terminates. The user sends the request and the server sends a successful or an unsuccessful terminate depending on whether or not he is successful in complying with the request.

Retrieve and Lookup requests involve data transfer from the server to the user. The user sends the request, the server responds with a rs, and transfers the data specified by the request. Upon completion of the data transfer, the server terminates the transaction sequence with a successful terminate if all goes well, or with an unsuccessful terminate is errors were detected.

Store and Append requests involve data transfer from the user to server. The user sends the request and the server

---

* The transactions (including terminate) are not "echoed", as echoing does not solve any "hung" conditions. Instead time-out mechanisms are recommended for avoiding hang-ups.

responds with a rr. The user then transfers the data. Upon receiving the data, the server terminates the sequence.

Execute request involves transfer of inputs from user to server, and transfer of outputs from server to user. The user sends the request to which the server responds with rr. The user then transfers the necessary inputs. The server "executes" the program or subroutine and tranfers the outputs to the user. Upon completion of the output transfer, the server terminates the transaction sequence.

3C. Aborts

Either host may abort the transaction sequence at any time by sending an unsuccessful terminate, or by closing the connection (NCP to transmit a CLS for the connection). The CLS is a more drastic type of abort and shall be used when there is a catastrophic failure or when an abort is desired in the middle of a long file transfer. The abort indicates to the receiving host that the other host wishes to terminate the transaction sequence and is now in the initial state. When CLS is used to abort, the using host will reopen the connection.

4. Data Types

4A. The data type code together with the extension code defines the manner in which the data field is to be parsed

and interpreted.* Although a large number of data types are defined, specific implementations may handle only a limited subset of data types. It is recommended that all host sites accept the "network ASCII" and "binary" data types. Host computers which do not "recognize" particular data types may abort the transaction sequence and return a data type error status code.

4B. The following data types are tentatively defined. The code in the type and extension field is represented by its ASCII equivalent with 8th bit as zero.

---

* The data type mechanism suggested here does not replace data reconfiguration service suggested by Harslem and Heafner in RFC 83 and NIC5772. In fact, it complements the reconfiguration. For example, data reconfiguration language can be expressed in EBCDIC, Network ASCII or any other code that form machine may "recognize". Subsequent data may be transmitted binary, and the form machine would reconfigure it to the required form. I have included in data types, a large number suggested by Harslem and Heafner, as I do not wish to preclude interpretation, reconfiguration and storage of simple forms of data at individual host sites.

Data Type	Code		
	Byte Size	Type	Extension
ASCII character,bit8=Ø(network)	8	A	NUL
ASCII characters, bit8=1	8	A	1
ASCII characters, bit8=even parity	8	A	E
ASCII characters, bit8=odd parity	8	A	O
ASCII characters, 8th bit info.	8	A	8
ASCII characters, 7 bits	7	A	7
ASCII characters, in 9-bit field	9	A	9
ASCII formatted files (with SOH, STX, ETX, etc.)	8	A	F
DEC-packed ASCII(5 7-bit char., 36th bit 1 or Ø)	36	A	D
EBCDIC characters	8	E	NUL
SIXBIT characters	6	S	NUL
Binary data	1	B	NUL
Binary bytes (size is binary ext.)	1-255	B	(any)
Decimal numbers, net ASCII	8	D	A
Decimal numbers, EBCDIC	8	D	E
Decimal numbers, sixbit	6	D	S
Decimal numbers, BCD(binary coded)	4	D	B
Octal numbers, net. ASCII	8	O	A
Octal numbers, EBCDIC	8	O	E
Octal numbers, SIXBIT	6	O	S
Hexadecimal numbers, net. ASCII	8	H	A
Hexadeciaml numbers, EBCDIC	8	H	E
Hexadecimal numbers, SIXBIT	6	H	S
Unsigned integers, binary (ext. field is byte size)	1-225	U	(any)
Sign magnitude integers (field is binary size)	1-255	I	(any)
2's complement integers (ext. field is byte size)	1-255	2	(any)
1's complement integers (ext. field is byte size)	1-255	1	(any)
Floating point (IBM36Ø)	32	F	I
Floating point (PDP-1Ø)	36	F	D
Status codes	8	S	NUL

4C. The data type information is intended to be interpretive. If a host accepts a data type, it can convert it to a form suited to its internal representation of characters or numbers.* Specifically when no conversion is to be performed, the data type used will be binary. The implicit or explicit byte size is useful as it facilitates storing of data. For example, if a PDP-1Ø receives data types A, A1, AE, or A7, it can store the ASCII characters five to a word (DEC-packed ASCII). If the datatype is A8 or A9, it would store the characters four to a word. Sixbit characters would be stored six to a word. If conversion routines are available on a system, the user of system program could convert the data from one form to another (such as EBCDIC to ASCII, IBM floating point to DEC floating point, Decimal ASCII to integers, etc.).

5. Initial Connection, CLS, and Identifying Users

5A. There will be a prearranged socket number** for the cooperating process on the serving host. The connection

---

* The internal character representation in the hosts may be different even in ASCII. For example PDP-1Ø stores 7-bit characters, five per word with 36th bit as don't care, while Multics stores them four per word, right-justified in 9-bit fields.

** It seems that socket 1 has been assigned to logger and socket 5 to NETRJS. Socket 3 seems a reasonable choice for the file transfer process.

establishment will be in accordance with the initial connection protocol of RFC 66 as modified by RFC 80. The NCP dialog would be:

user to server: RTS<us><3><p>

if accepted, server to user: STR<3><us><CLS><3><us>

server to user on link p: <ss>

server to user: STR<ss+1><us>RTS<ss><us+1><q>

user to server: STR<us><ss+1>RTS<us+1><ss><r>

This sets up a full-duplex connection between user and server processes, with server receiving through local socket ss from remote socket us+1 via link q, and sending to remote socket us through local socket ss+1 via link r.

5B. The connection will be broken by trading a CLS between the NCP'S for each of the two connections. Normally the user will initiate the CLS.

CLS may also be used by either the user or the server to abort a data transmission in the middle. If a CLS is received in the middle of a transaction sequence, the whole transaction sequence will be aborted. The using host will then reopen the connection.

5C. The first transaction from the user to server will be the identify transaction. The users will be identified by the pathname in data field of the transaction which should be a form acceptable to the server. The server is at

liberty to truncate pathnames for its own use. Since the identify transaction does not require a response or terminate, the user can proceed directly with other requests.

IV. Extensions to Protocol

The protocol specified above has been designed to be extendable. The obvious extensions would be in the area of transaction types (new types of requests), error return status words, and data types. Some of the non-obvious extensions, that I can visualize are provisions of access control mechanisms, developing a uniform way of specifying file attributes in headings of files, increasing the scope of the execute command to include subroutine mediation, and the provision of transaction sequence identification numbers to facilitate handling of multiple requests over the same connection pair.

Users of protected file systems should be able to have a reasonable degree of confidence in the ability of the serving process to identify remote users correctly. In the absence of such confidence, some users would not be willing to give access to the serving process (especially write access). Inclusion of access control mechanisms such as passwords, is likely to enhance the indirect use of network by users who are concerned about privacy and security. A simple extension to the protocol would be to have the serving host sent a transaction type "password?" after it receives user name. Upon receipt of "password?" the

using host will transmit the password, which when successfully acknowledged, would indicate to the user that requests may proceed.

There are a number of file attributes which properly belong in the heading of a file rather than the file itself or the data type in descriptors of transactions. Such attributes include access control lists, date file was last modified, information about the nature of file, and the description of its contents in a data description or data reconfiguration language. Some uniformity in the way file attributes are specified would be useful. Until then, the interpretation of the heading would be up to the user or the using process. For example, the heading of files which are input to a data reconfiguration (form) machine may be the desired transformations expressed in the reconfiguration language.

The "execute" command which achieves the execution of programs resident in remote hosts is a vital part of indirect use of remote hosts. The present scope of the execute command, as outlined in the specifications, is somewhat limited. It assumes that the user or using process is aware of the manner in which the arguments and results should be exchanged. One could broaden the scope of execute command by introducing a program mediation protocol.*

The present specification of the protocol does not allow the simultaneous transfer and processing of multiple requests over

---

* The term program mediation was suggested by Bob Metcalfe who is intending to write a paper on this subject.

the same pair of connections. If such a capability is desired, there is an easy way to implement it which only involves a minor change. A transaction sequence identification number (TSid) could replace a NUL field in the descriptor of transactions. The TSid would facilitate the coordination of transactions, related to a particular transaction sequence. The 256 code combinations permitted by the TSid would be used in a round-robin manner (I can't see more than 256 outstanding requests between two user-processes in any practical implementation). An alternate way of simultaneous processing of requests is to open new pairs of connection. I am not sure as to how useful simultaneous processing of requests is, and which of the two is a more reasonable approach.

V. Conclusions

I have tried to present a user-level protocol that will permit users and using programs to make indirect use of remote host computers. The protocol facilitates not only file system operations but also program execution in remote hosts. This is achieved by defining requests which are handled by cooperating processes. The transaction sequence orientation provides greater assurance and would facilitate error control. The notion of data types is introduced to facilitate the interpretation, reconfiguration and storage of simple and limited forms of data at individual host sites. The protocol is readily extendable.